11 | 存储系统：数据到底都存哪儿了？

2021-10-04 吴磊

《零基础入门Spark》 课程介绍 >

讲述：吴磊

时长 14:52 大小 13.62M

你好，我是吴磊。

感谢你在国庆假期仍然坚持学习，今天这一讲，我们来学习存储系统，与调度系统一样，它也是 Spark 重要的基础设施之一。不过，你可能会好奇：“掌握 Spark 应用开发，需要去了解这么底层的知识吗？”坦白地说，还真需要，为什么这么说呢？

我们前面学了 Shuffle 管理、RDD Cache 和广播变量，这些功能与特性，对 Spark 作业的执行性能有着至关重要的影响。而想要实现这些功能，底层的支撑系统正是 Spark 存储系统。

学习和熟悉存储系统，不单单是为了完善我们的知识体系，它还能直接帮你更好地利用 RDD Cache 和广播变量这些特性。在未来，这些知识也能为你做 Shuffle 的调优奠定良好

的基础。

既然存储系统这么重要，那要怎样高效快速地掌握它呢？本着学以致用的原则，我们需要先了解系统的服务对象，说白了就是存储系统是用来存什么东西的。

服务对象

笼统地说，**Spark 存储系统负责维护所有暂存在内存与磁盘中的数据，这些数据包括 Shuffle 中间文件、RDD Cache 以及广播变量**。

对于上述三类数据，我们并不陌生。我们先回顾一下什么是 Shuffle 中间文件，在 Shuffle 的计算过程中，Map Task 在 Shuffle Write 阶段生产 data 与 index 文件。接下来，根据 index 文件提供的分区索引，Shuffle Read 阶段的 Reduce Task 从不同节点拉取属于自己的分区数据。而 Shuffle 中间文件，指的正是两个阶段为了完成数据交换所仰仗的 data 与 index 文件。

RDD Cache 指的是分布式数据集在内存或是磁盘中的物化，它往往有利于提升计算效率。广播变量上一讲我们刚刚介绍过，它的优势在于以 Executors 为粒度分发共享变量，从而大幅削减数据分发引入的网络与存储开销。

我们刚才对这三类数据做了简单回顾，如果你觉得哪里不是特别清楚的话，不妨翻回前面几讲再看一看，我们在第 7、8、10 这 3 讲分别对它们做了详细讲解。好啦，了解了存储系统服务的主要对象以后，接下来，我们来细数 Spark 存储系统都有哪些重要组件，看看它们之间又是如何协作的。

存储系统的构成

理论的学习总是枯燥而又乏味，为了让你更加轻松地掌握存储系统的核心组件，咱们不妨还是用斯巴克国际建筑集团的类比，来讲解 Spark 存储系统。

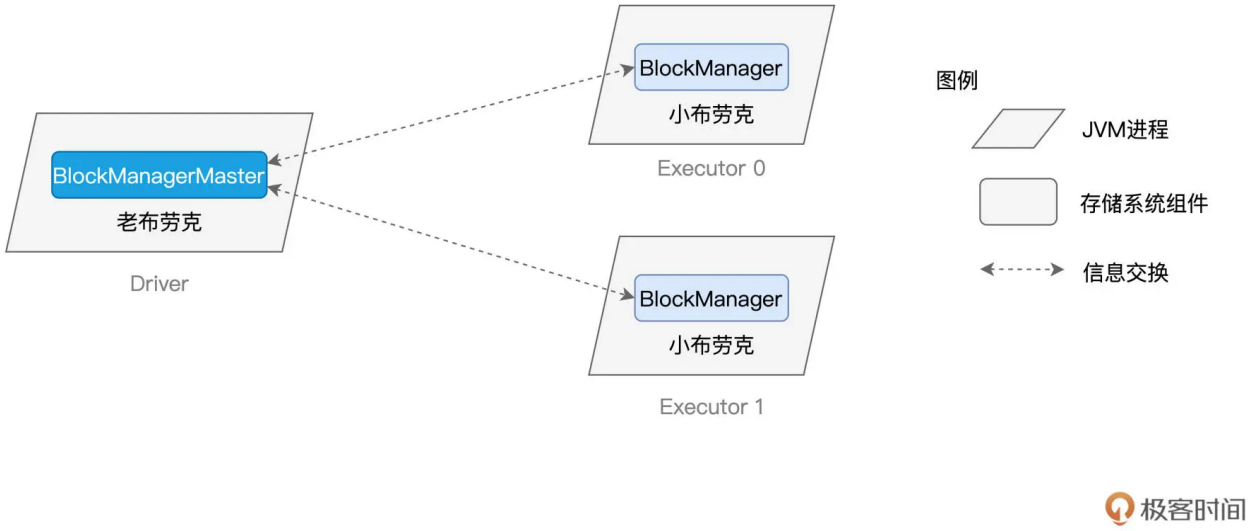
相比调度系统复杂的人事关系（戴格、塔斯克、拜肯德），存储系统的人员构成要简单得多。在内存管理那一讲，我们把节点内存看作是施工工地，而把节点磁盘看作是临时仓库，那么显然，管理数据存储的组件，就可以看成是仓库管理员，简称库管。

布劳克家族

在斯巴克建筑集团，库管这个关键角色，一直以来都是由布劳克家族把持着。

布劳克家族在斯巴克集团的地位举足轻重，老布劳克（BlockManagerMaster）坐镇集团总公司（Driver），而他的子嗣们、小布劳克（BlockManager）则驻守在各个分公司（Executors）。

对集团公司建材与仓库的整体情况，老布劳克了如指掌，当然，这一切要归功于他众多的子嗣们。各家分公司的小布劳克，争先恐后地向老爸汇报分公司的建材状态与仓库状况。关于他们的父子关系，我整理到了下面的示意图中。



存储系统：布劳克家族

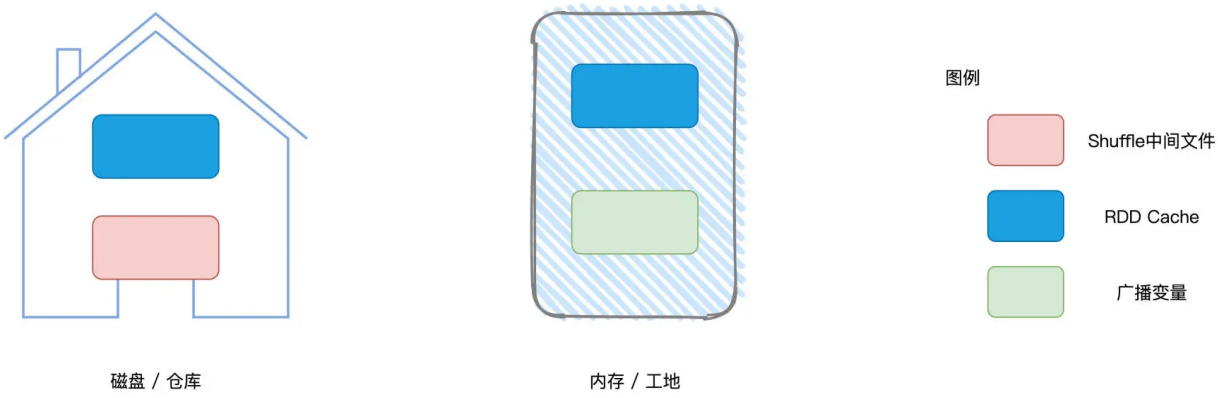
从上图我们能够看得出来，小布劳克与老布劳克之间的信息交换是双向的。不难发现，布劳克家族的家风是典型的“家长制”和“一言堂”。如果小布劳克需要获取其他分公司的状态，他必须要通过老布劳克才能拿到这些信息。

在前面的几讲中，我们把建材比作是分布式数据集，那么，BlockManagerMaster 与 BlockManager 之间交换的信息，实际上就是 Executors 之上数据的状态。说到这里，你可能会问：“既然 BlockManagerMaster 的信息都来自于 BlockManager，那么 BlockManager 又是从哪里获取到这些信息的呢？”要回答这个问题，我们还要从 BlockManager 的职责说起。

我们开头说过，存储系统的服务对象有 3 个：分别是 Shuffle 中间文件、RDD Cache 以及广播变量，而 BlockManager 的职责，正是在 Executors 中管理这 3 类数据的存储、读写

与收发。就存储介质来说，这 3 类数据所消耗的硬件资源各不相同。

具体来说，Shuffle 中间文件消耗的是节点磁盘，而广播变量主要占用节点的内存空间，RDD Cache 则是“脚踏两条船”，既可以消耗内存，也可以消耗磁盘。



极客时间

服务对象与存储介质

不管是在内存、还是在磁盘，这些数据都是以数据块（Blocks）为粒度进行存取与访问的。**数据块的概念与 RDD 数据分区（Partitions）是一致的**，在 RDD 的上下文中，说到数据划分的粒度，我们往往把一份数据称作“数据分区”。而在存储系统的上下文中，对于细分的一份数据，我们称之为数据块。

有了数据块的概念，我们就可以进一步细化 BlockManager 的职责。**BlockManager 的核心职责，在于管理数据块的元数据（Meta data），这些元数据记录并维护数据块的地址、位置、尺寸以及状态**。为了让你直观地感受一下元数据，我把它的样例放到了下面的示意图里，你可以看一看。

存储介质	地址	位置	大小	状态
内存	Host0:BlockManager0	内存地址: 0xF38A02EB	128MB	可用
磁盘	Host1:BlockManager1	文件系统路径: /XXX/blockX	128MB	可用

极客时间

元数据样例

只有借助元数据，BlockManager 才有可能高效地完成数据的存与取、收与发。这就回答了前面我提出的问题，BlockManager 与数据状态有关的所有信息，全部来自于元数据的管理。那么接下来的问题是，结合这些元数据，BlockManager 如何完成数据的存取呢？

不管是工地上，还是仓库里，这些场所都是尘土飞扬、人来人往，像存取建材这种事情，养尊处优的小布劳克自然不会亲力亲为。于是，他招募了两个帮手，来帮他打理这些脏活累活。

这两个帮手也都不是外人，一个是大表姐迈美瑞（MemoryStore），另一个是大表哥迪斯克（DiskStore）。顾名思义，**MemoryStore 负责内存中的数据存取，而相应地，DiskStore 则负责磁盘中的数据访问**。

好啦，到此为止，存储系统的重要角色已经悉数登场，我把他们整理到了下面的表格中。接下来，我们以 RDD Cache 和 Shuffle 中间文件的存取为例，分别说一说迈美瑞和迪斯克是如何帮助小布劳克来打理数据的。

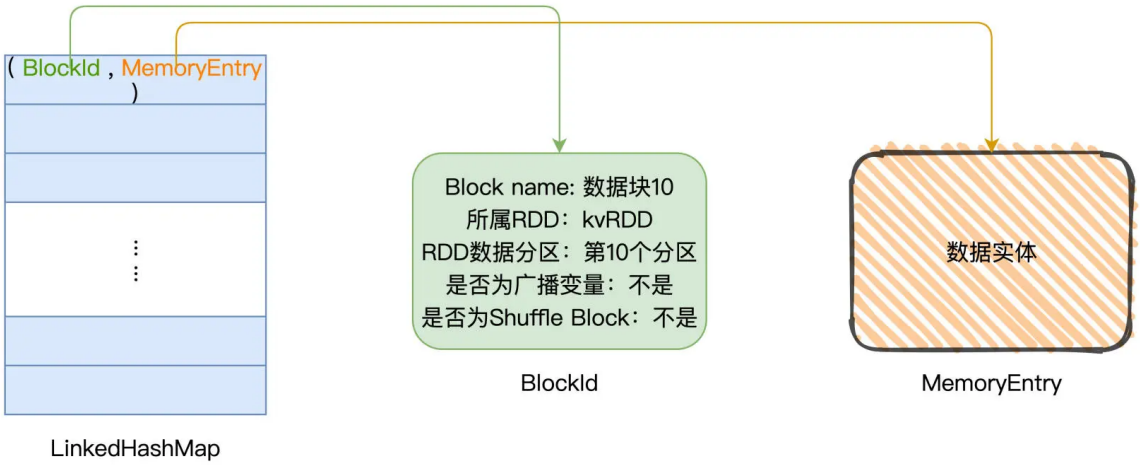
类比对象	主要角色			
Spark	BlockManagerMaster	BlockManager	MemoryStore	DiskStore
斯巴克集团	老布劳克	小布劳克	迈美瑞	迪斯克

极客时间

存储系统主要角色

MemoryStore：内存数据访问

大表姐迈美瑞秀外慧中，做起事情来井井有条。为了不辜负小布劳克的托付，迈美瑞随身携带着一本小册子，这本小册子密密麻麻，记满了关于数据块的详细信息。这个小册子，是一种特别的数据结构：LinkedHashMap[BlockId, MemoryEntry]。顾名思义，LinkedHashMap 是一种 Map，其中键值对的 Key 是 BlockId，Value 是 MemoryEntry。



极客时间

迈美瑞的小册子：LinkedHashMap

BlockId 用于标记 Block 的身份，需要注意的是，BlockId 不是一个仅仅记录 Id 的字符串，而是一种记录 Block 元信息的数据结构。BlockId 这个数据结构记录的信息非常丰富，包括 Block 名字、所属 RDD、Block 对应的 RDD 数据分区、是否为广播变量、是否为 Shuffle Block，等等。

MemoryEntry 是对象，它用于承载数据实体，数据实体可以是某个 RDD 的数据分区，也可以是广播变量。存储在 LinkedHashMap 当中的 MemoryEntry，相当于是通往数据实体的地址。

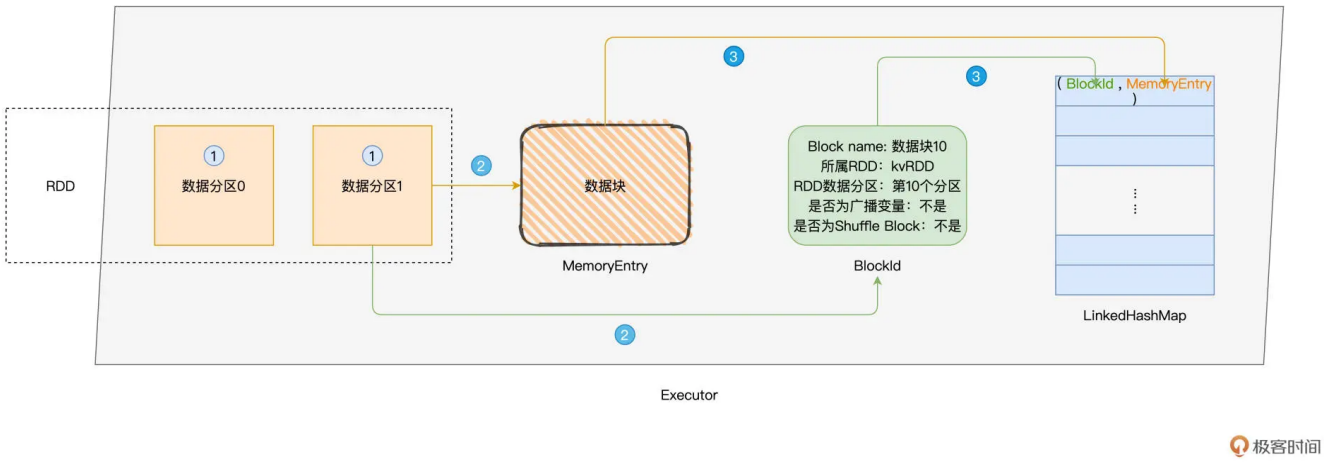
不难发现，BlockId 和 MemoryEntry 一起，就像是居民户口簿一样，完整地记录了存取某个数据块所需的所有元信息，相当于“居民姓名”、“所属派出所”、“家庭住址”等信息。**基于这些元信息，我们就可以像“查户口”一样，有的放矢、精准定向地对数据块进行存取访问。**

复制代码

```
1 val rdd: RDD[_] = _
2 rdd.cache
3 rdd.count
```

以 RDD Cache 为例，当我们使用上述代码创建 RDD 缓存的时候，Spark 会在后台帮我们做如下 3 件事情，这个过程我把它整理到了下面的示意图中，你可以看一看。

1. 以数据分区为粒度，计算 RDD 执行结果，生成对应的数据块；
2. 将数据块封装到 MemoryEntry，同时创建数据块元数据 BlockId；
3. 将（BlockId，MemoryEntry）键值对添加到“小册子”LinkedHashMap。



RDD Cache的计算过程

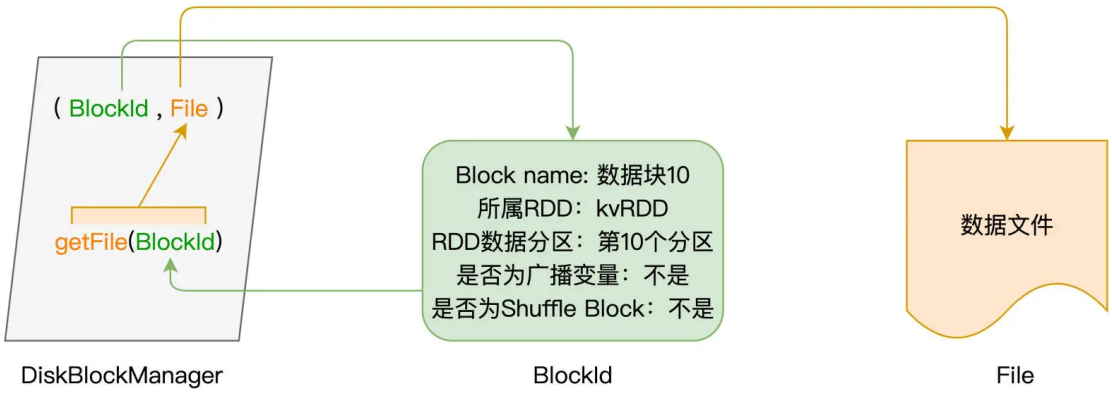
随着 RDD Cache 过程的推进，LinkedHashMap 当中的元素会越积越多，当迈美瑞的小册子完成记录的时候，Spark 就可以通过册子上的“户口簿”来访问每一个数据块，从而实现对 RDD Cache 的读取与访问。

DiskStore：磁盘数据访问

说完大表姐，接下来，我们再来说说大表哥迪斯克。迪斯克的主要职责，是通过维护数据块与磁盘文件的对应关系，实现磁盘数据的存取访问。相比大表姐的一丝不苟、亲力亲为，迪斯克要“鸡贼”得多，他跟布劳克一样，都是甩手掌柜。

看到大表姐没日没夜地盯着自己的“小册子”，迪斯克可不想无脑地给布劳克卖命，于是他招募了一个帮手：DiskBlockManager，来帮他维护元数据。

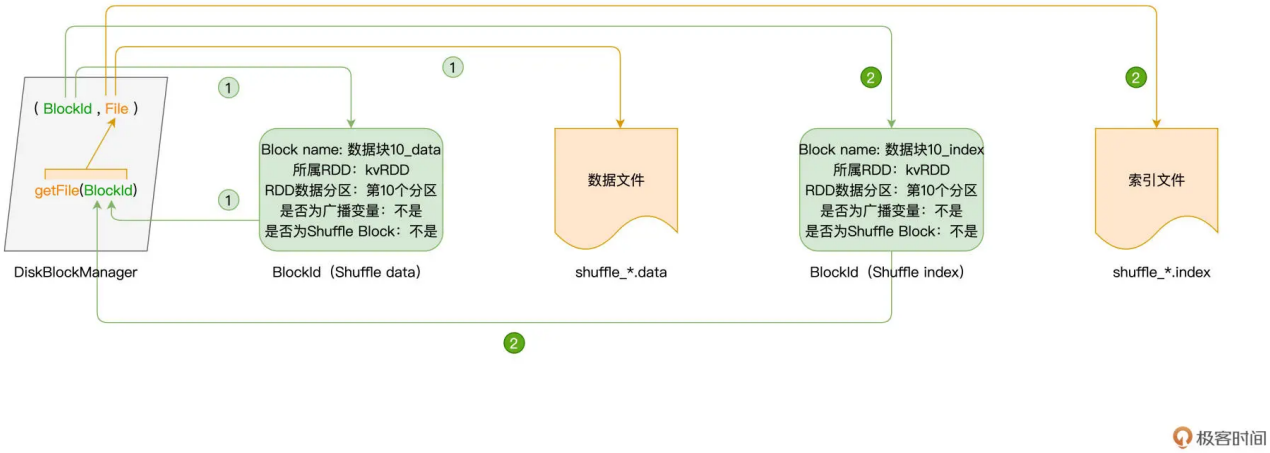
有了 DiskBlockManager 这个帮手给他打理各种杂事，迪斯克这个家伙就可以哼着小曲、喝着咖啡，坐在仓库门口接待来来往往的施工工人就好了。这些工人有的存货，有的取货，但不论是干什么的，迪斯克会统一把他们打发到 DiskBlockManager 那里去，让 DiskBlockManager 告诉他们货物都存在哪些货架的第几层。



迪斯克的帮手：DiskBlockManager

帮手 DiskBlockManager 是类对象，它的 getFile 方法以 BlockId 为参数，返回磁盘文件。换句话说，给定数据块，要想知道它存在了哪个磁盘文件，需要调用 getFile 方法得到答案。**有了数据块与文件之间的映射关系，我们就可以轻松地完成磁盘中的数据访问。**

以 Shuffle 为例，在 Shuffle Write 阶段，每个 Task 都会生成一份中间文件，每一份中间文件都包括带有 data 后缀的数据文件，以及带着 index 后缀的索引文件。那么对于每一份文件来说，我们都可以通过 DiskBlockManager 的 getFile 方法，来获取到对应的磁盘文件，如下图所示。



根据BlockId获取Shuffle中间文件

可以看到，获取 data 文件与获取 index 文件的流程是完全一致的，他们都是使用 BlockId 来调用 getFile 方法，从而完成数据访问。

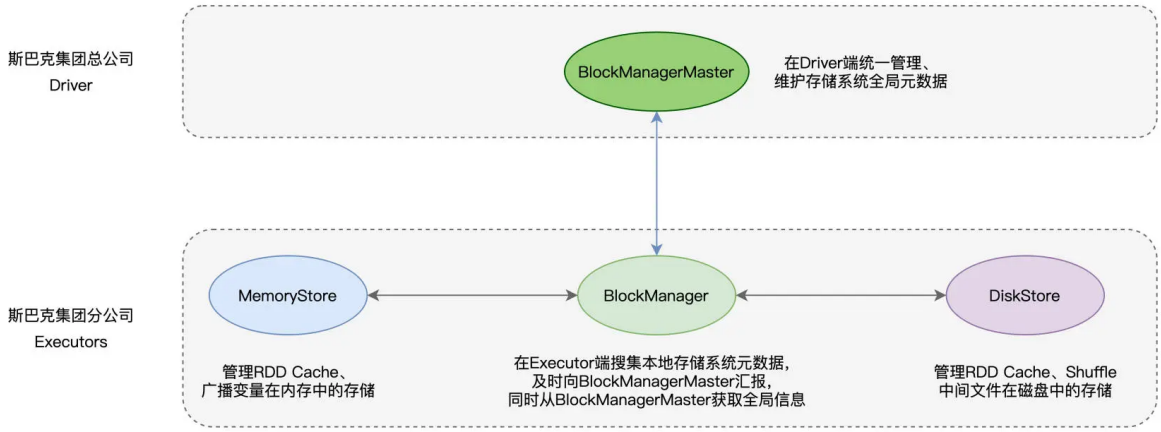
重点回顾

今天这一讲，我们重点讲解了 Spark 存储系统。关于存储系统，你首先需要知道是，RDD Cache、Shuffle 中间文件与广播变量这三类数据，是存储系统最主要的服务对象。

接着，我们介绍了存储系统的核心组件，它们是坐落在 Driver 端的 BlockManagerMaster，以及“驻守”在 Executors 的 BlockManager、MemoryStore 和 DiskStore。BlockManagerMaster 与众多 BlockManager 之间通过心跳来完成信息交换，这些信息包括数据块的地址、位置、大小和状态，等等。

在 Executors 中，BlockManager 通过 MemoryStore 来完成内存的数据存取。MemoryStore 通过一种特殊的数据结构：LinkedHashMap 来完成 BlockId 到 MemoryEntry 的映射。其中，BlockId 记录着数据块的元数据，而 MemoryEntry 则用于封装数据实体。

与此同时，BlockManager 通过 DiskStore 来实现磁盘数据的存取与访问。DiskStore 并不直接维护元数据列表，而是通过 DiskBlockManager 这个对象，来完成从数据库到磁盘文件的映射，进而完成数据访问。



极客时间

每课一练

LinkedHashMap 是一种很特殊的数据结构，在今天这一讲，我们仅介绍了它在 Map 方面的功用。你可以试着自己梳理一下 LinkedHashMap 这种数据结构的特点与特性。

期待在留言区看到你的思考。如果这一讲对你有帮助，也推荐你转发给更多的同事、朋友。我们下一讲见！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 10 | 广播变量 & 累加器：共享变量是用来做什么的？

下一篇 12 | 基础配置详解：有哪些配置项是你必须要关注的？

精选留言 (3)

写留言

Unknown element
2021-10-06

老师在文章倒数第二张图片里shuffle中间文件的blockid里有个属性叫 是否为shuffle block，这里的shuffle block是什么呢？为什么shuffle中间文件不属于shuffle block?

Neo-dqy

2021-10-05

【BlockManagerMaster 与众多 BlockManager 之间通过**心跳**来完成信息交换】，可以问一下老师这个心跳机制是什么呀？具体是怎么实现的呢？

展开

qinsi

2021-10-04

LinkedHashMap 可用来实现 LRU Cache

展开